AFRL-IF-RS-TR-1998-235
Final Technical Report
January 1999

POLICY SPECIFICATION LANGUAGE DESIGN

Odyssey Research Associates, Inc.

David Rosenthal, Matt Stillerman, and Roshan Thomas

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-1998-235 has been reviewed and is approved for publication.

APPROVED: [illegible]

JOSEPH V. GIORDANO
Project Engineer

FOR THE DIRECTOR: [illegible]

WARREN H. DEBANY JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** January 1999	**3. REPORT TYPE AND DATES COVERED** Final Jun 96 - Oct 97

4. TITLE AND SUBTITLE POLICY SPECIFICATION LANGUAGE DESIGN	**5. FUNDING NUMBERS** C - F30602-96-C-0213 PE - 33401G PR - 1069 TA - 01 WU - 04
6. AUTHOR(S) David Rosenthal, Matt Stillerman, and Roshan Thomas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Odyssey Research Associates, Inc. Cornell Business and Technology Park 33 Thornwood Drive, Suite 500 Ithaca NY 14850-1250	**8. PERFORMING ORGANIZATION REPORT NUMBER** ORA-TM-97-0029
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGB 525 Brooks Road Rome NY 13441-4505	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-1998-235

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Joseph V. Giordano/IFGB/(315) 330-4199

12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

The purpose of this investigation was to aid developers of groupware applications, such as applications using Lotus Notes, in choosing appropriate security controls.

Products such as Lotus Notes have security controls (e.g., access control lists, encrypted sections, and digital signatures) for building applications that meet complex security policies. It may be difficult for the application developer to select the right combination of controls to meet the desired security policy.

This report describes a security policy language that can express general policy constraints on users and data, as well as constraints that directly map to Lotus Notes security controls. It can capture both the type of protection desired as well as some aspects of the assurance level.

14. SUBJECT TERMS Computer Security, Security Policy, Access Control	**15. NUMBER OF PAGES** 72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Abstract

The purpose of this investigation was to aid developers of groupware applications, such as applications using Lotus Notes, in choosing appropriate security controls.

Products such as Lotus Notes have security controls (e.g., access control lists, encrypted sections, and digital signatures) for building applications that meet complex security policies. It may be difficult for the application developer to select the right combination of controls to meet the desired security policy.

This report describes a security policy language that can express general policy constraints on users and data, as well as constraints that directly map to Lotus Notes security controls. It can capture both the type of protection desired as well as some aspects of the assurance level.

Table Of Contents

List of Figures

1. Summary

The purpose of this investigation was to design a policy language for Lotus Notes that would aid application developers in choosing appropriate security controls. Lotus Notes has security controls for building applications that meet complex security policies. It can be difficult for the application developer to select the right combination of controls to meet the desired security policy.

This effort developed a policy language that expresses the general policy constraints on data and the constraints that directly map to application level security controls. This method permits capturing the type of protection that is desired, as well as some aspects about the assurance required for that protection.

We believe that the results of this effort will provide significant benefit in aiding developers in selecting the right security for their applications.

To facilitate the use of this policy language, it would be useful to augment it with tool support for capturing and analyzing security specifications.

2. Introduction

When an organization automates document handling with software such as Lotus Notes, the security mechanisms of the software enforce some security policy. Policymakers are confronted with two kinds of security questions:

- Is the access adequate? Do people (and servers) have the access required to fulfill their roles?
- Is protection adequate? Are the privacy and integrity of the organization's data ensured with sufficient strength?

An implementor must find a technical solution that balances these objectives.

Developing the security enforcement for Lotus Notes applications is also difficult because there are frequently several ways to meet specific policy objectives. These enforcement mechanisms may differ in the degree of assurance that they provide. The choices an implementor makes often interact with one another, possibly producing unexpected results. Also, it is easy for an implementor to forget to enforce something. Hence, the construction and use of a specification is likely to be of real benefit. This report describes a language in which the specification can be expressed.

Developers can use the specification language to express the intended access requirements, as well as certain aspects of the implementation. The language emphasis is on handling common access control policies for application developers who are not security experts.

Although not part of this effort, there is the potential for tool support to advise developers about problems in the policy or implementation approach. There is also the potential for tools that provide a user interface to help in building the specification. Ideally, an application developer would use the interface and not the language in constructing the specification. These topics are briefly covered in this report.

The presentation for the report is organized as follows:

- We begin with some background to Lotus Notes and its security capabilities.
- We describe an overview of the policy language.
- The semantics and the syntax of the language are then presented.
- The language is illustrated with an example.
- Possible enhancements of Lotus Notes security mechanisms are described.
- Finally, we present our conclusions, followed by our recommendations.

3. Background

3.1 Lotus Notes

3.1.1 Lotus Notes Introduction

Lotus Notes manages the collection and flow of information. Here are some typical business uses (from the Lotus Notes Application Developer's Guide):

- Approvals — Purchasing Requests, Expense Reports, Marketing Plans, Loan Approvals, Insurance Claims
- Broadcasting — Industry News, Meeting Agendas and Minutes, News letters
- Discussions — Brainstorming database, Feedback or Opinion database, Suggestion Box
- Reference — Policies and Procedures Handbook, Market Research Reference, Software Code Library, Customer Support Information, Compensation and Benefits, Employee Reviews, Address Book
- Tracking — Event Planning Database, Sales Account Tracking, Resume tracking, Project Status, Service Tracking, Inventory Supply, Credit and Collections

Lotus Notes runs on many types of machines, including Windows, OS/2, UNIX, and Macintosh. It allows multiple users to access the same data and it permits access to information though network or modem. It utilizes electronic mail to distribute information and this capability can be used to set up simple automatic workflows, such as automatic document routing or tracking business processes. It supports embedded objects such as spreadsheets.

This document describes only Lotus Notes version 4.1. Recently, there have been some significant enhancements, particularly with respect to the Domino/Lotus web browser. Third party products can also be used to extend Lotus Notes security capabilities.

3.1.2 Some Terminology

In this section, we define some basic Lotus Notes terminology.

- **Server:** Lotus Notes servers are processes that "own" data and that handle requests concerning that data for users and other servers. Users may also have direct access to some data, so-called "local" data.
- **Database:** A database is principally a dynamic collection of data objects, called documents (see the next definition). Typically, documents may be created or destroyed by certain users or in response to certain events. Additionally, databases contain their own "design," including

  - ♦ forms, fields, etc.;
  - ♦ views, navigators, and folders;
  - ♦ access controls; and
  - ♦ scripts, agents, etc.

  Sufficiently privileged users may change the design part of a database.

  A server will typically serve-up several databases. A user may have access to one or more local databases as well as those available through servers.

  A *replica* is a database that is a synchronized copy of some other database. Replicas can be resynchronized either upon command or with scheduled updates. A replica can be local or it can reside on a server.

- **Document:** Documents are the principal objects of a database. The structure of a document is given by a *form* (see the next definition). We say that the document is an *instance* of the form.

  When a document is retrieved and viewed by a Lotus Notes client, the user sees a scrollable window containing the data of that document (e.g., text or numbers). Some data may be hidden and some may be in expandable sections. Fonts, colors, graphics, and the arrangement of data on the page may be specified. Documents may also contain OLE data from other applications (e.g., spreadsheets or graphics) and hypertext links to other Lotus documents and databases.

- **Form:** A form is a part of a database that describes a *uniform* collection of documents. It describes (and controls) what kinds of data make up those documents (the fields and their types), and the visual appearance of the documents. Every document is an instance of a form. A database may have several forms. These forms are part of the "design" of the database.

A sufficiently privileged user may access a form through the Lotus Notes client. When a user does so, forms appear as special documents in the database. They may be read, edited, created, deleted, and access controlled, as with an ordinary document.

- **Field:** Documents contain fields. A field is a named, typed datum. The available types (e.g., text or integer) are fixed by Lotus. As mentioned above, the fields that make up a document are determined by the form of the document. The values of those fields are (typically) set by the creator of the document. Some fields may have values that are computed rather than being set "by hand."

  Fields may be directly part of a document, or they may be contained in *sections*. The name, visibility, appearance, and type of a field in a document are governed by the form of the document, and will be the same for all documents of that form.

  A field may be *encryption enabled.* A field may be *signed enabled.* A section may be *access controlled.* The meaning of these flags is discussed in Section 3.2.6.

- **Folder, View, Navigator:** These are special "windows" of information, which are defined by a database and which summarize the contents of that database. They are typically used to help users find particular documents. A view is a table of the documents in the database. It may be configured to show only some documents, to show particular information about each document, and to be sorted in specific ways. A user may click on an entry in a view and thus open the corresponding document.

  A folder is very much like a view, except that the documents to be included in the folder are selected by hand, while the contents of a view are computed.

  A navigator is a graphical (iconic) representation of the contents of a database. It also provides mouse-click access to documents.

- **Agent:** An agent is a programmed action or process, which may be scheduled regularly, or may be initiated "by hand."

- **Certificate or ID:** A certificate is cryptographic data which, when presented to a Lotus server, establishes the presenter's identity. It is sometimes referred to as an "ID." This data is usually protected by a password, which the presenter will be required to enter. Servers also have certificates (and thus identities).

  A certificate contains a public-private key pair, which allows the holder to sign documents and which allows him to read those documents which have been encrypted "for him" by others with his public key. This latter method is how e-mail is typically protected.

  Certificates are issued (and signed) by certifiers, each of which has a special certificate.

All certificates contain the entire ancestry of certifiers, which are responsible for this certificate. This is a sequence of "begets" going back to an initial "Organizational Certifier".

Certificates may contain additional secret encryption keys. They may also contain cross-certifications.

- **Domain:** Each certificate or ID has a unique name, which consists of its local name, followed by the names of its ancestor certifiers separated by slashes. Thus, the space of names with a common ancestor is hierarchical or tree-like. The space of all names that descend from an Organizational Certifier is called a domain.

  Names within a domain are mutually intelligible (and trusted). Thus, a server will recognize the ID of a user if the server's ID is in the same domain. Generally, a server will not accept an ID from a different domain than its own. However, such a foreign ID may be cross-certified by the server's domain, and thus become acceptable.

- **Group:** A group is a named collection of IDs. It is defined and maintained in the Public Address Book (PAB), a special Lotus Notes database. Access control lists may grant access to groups, which causes all members of the group to have that access.

- **Role:** Each database may have defined several roles. These are like groups, but are local to the database. An entry in the database's ACL can be assigned some of those roles (or none). Access control lists for individual documents and parts of documents can also have entries that name roles. Other entries in those ACLs can still refer to individual IDs and groups. (Note that ACLs can also locally introduce roles.)

3.2 Lotus Notes Security

In this section, we describe a summary of Lotus Notes security features. The main purpose of this presentation is to provide sufficient background of Lotus security so that the security policy language is comprehensible. Additionally, we highlight some security concerns associated with these features that could be used to help assess the security of a specified system. While this information will be useful for security analysis, it is not intended as comprehensive product security profile.

We present the Lotus security features according to what part of the system architecture they are associated with. This decomposition consists of the following architectural layers:

- server,
- database,
- document (or form), and
- field or section.

To access a particular field of a particular document in a particular database on a particular server, the user must have appropriate kinds of access at all of these layers.

3.2.1 Server Security

A basic part of the security of a Lotus application is the physical protection of the server machines. This includes limitations on remote access, such as telnet, ftp, http, and PCAnywhere, as well as limitations on direct access to the machines by individuals. (Our analysis is based on Lotus Notes 4.1. Newer versions of Lotus Notes can use a Domino Web server, for which there are associated network security issues.)

Servers running on NT and UNIX should be able to rely, in part, on file system access controls (file ownership and permissions) to maintain the isolation of the Notes data. The intended Lotus usage is that users should be able to access Lotus data only via the Notes server, never directly or with other programs. (As data may be replicated on a local workstation, file system access and physical protection may also be a security consideration for platforms that are not servers.)

Each domain has an associated Notes database, called the Public Address Book, which describes the servers, users, networks, and so forth, of the domain. This database serves to control the operation of the resources in the domain. Hence, the security for the Public Address Book is particularly important.

The Public Address Book document describing a server allows the following items to be controlled:

- Identity of the administrator of the server.
- Whether to compare public keys against those in the address book.
- Whether to allow anonymous connections.
- Whether to allow access only to users listed in the address book (in separate user documents), or whether to allow anyone with a certifier in common with the server to have access. Thus, this feature can turn the address book into a kind of server ACL.
- An "Access Server" list and a "Not Access Server" list.
- A list of who can create new databases on the server.
- A list of who can create new replica databases on the server.
- A list of who can use this server for pass-through access to other servers, and what they are permitted to do.
- A list of who can run personal agents on the server.
- A list of who can run restricted Lotus Script agents on the server.
- A list of who can run unrestricted Lotus Script agents on the server.

It should be noted that the Public Address Book database and its documents are all subject to access control, like any others. In a typical installation, most users would not be able to edit the Public Address Book.

In addition to security for the Public Address Book, a Notes server must maintain security for all other Lotus files on the server associated with application support:

- database files,
- templates for databases, and
- item IDs (see section 3.2.1.1).

3.2.1.1 Identities

Lotus provides users and servers with signed certificates or "IDs". Users and servers use IDs to authenticate one another as part of access control, to sign documents, and to encrypt data for privacy. IDs are described in more detail in section 3.1.2.

3.2.1.2 Connections

A connection is a document in the Public Address Book that governs how servers interact with one another. These are used to enable and schedule mail routing and database replication.

3.2.2 Database Security

3.2.2.1 Controlling Database Security

Database security can be controlled in a number of ways:

Location of Database

To have access to a database, a user (or server) must first have access to the server on which the database is located. If the database is local, then the user must have access to the machine on which it resides (the database could be encrypted with the user's key).

Directory of the Database

On machines with file/directory access controls, such as UNIX and NT, one can place a database in a protected directory.

Applying Encryption to a Database

If a user creates a local database or replica, then the user's key can encrypt that file. Only that user can have any access to that database. If the database or replica resides on a server, then the server's key encrypts the file. All access to the server database must then be through that server. Encryption for a database may be "simple", "medium", or "strong".

Visibility of Database Existence

Creators can control whether a database should be listed in the database library, and whether it should be shown in the "open database" dialog. This is not a real "security" feature, since Lotus Script programs can be written to open databases which have either of these characteristics. However, this may stop a casual user, especially if the name and other particulars of the database are not publicized.

Access Control List (ACL) for Databases

One of the most important security services for databases is an ACL. This service is described in the section below.

3.2.2.2 Access Control Lists for Databases

Associated with every database is an access control list (ACL). An ACL controls the kind of access permitted to the database for users and servers. Note that this refines, but cannot override, any access limitations imposed on an identity at the server layer.

Access of an identity to a database falls into one of seven "levels," ranging from *No Access* to *Manager*. Within each access level, there may be several optional capabilities, which can be individually turned on or off. Here is a summary of the main capabilities of each access level.

- **No Access:** Cannot even open the database.
- **Depositor:** Can open the database and create documents. Cannot read any documents.
- **Reader:** Can open the database and (generally) read documents. Cannot create or edit documents.
- **Author:** Can modify documents that the user created and can read other documents. The creator of a document may be able to list others as authors, in which case, if they had Author access, they could also modify that document.
- **Editor:** Can modify or read any document.
- **Designer:** Can modify or read any document. Can change the design of the database. This is a very privileged level, since many security controls can be circumvented by a Designer.
- **Manager:** Can modify or read any document. Can change the design of the database. Can change the ACL of the database.

Figure 1 shows the capabilities that can be separately administered for a user within each access level. In this table, R means that the capability is Required at that level, D means that it is Denied and S means that it is Settable.

```
                Create Documents
                | Delete Documents
                | | Create Personal Agents
                | | | Create Personal Folders/Views
                | | | | Create Shared Folders/Views
                | | | | | Create Lotus Script Agents
LEVELS:         | | | | | |
Manager         R S R R R R
Designer        R S R R R R
Editor          R S S S S S
Author          S S S S D S
Reader          D D S S D S
Depositor       R D D D D D
No Access       D D D D D D
```

Figure 1: Capabilities

For instance, a Reader may be able to create Personal Agents, but cannot delete documents.

The ACL may also define one or more "roles." A role is an identifier that may be used to control fine-grained access to documents and fields in that database.

Each entry in the ACL

- names an identity or group of identities, or "default",
- specifies the kind of entity: unspecified, user, server, user-group, server-group, mixed-group,
- gives an access level (one of the seven above)
- turns on or off each of the settable privileges for the access level (see Figure 1), and
- assigns zero or more roles to the identity.

Groups are collections of identities that are defined by special documents in the Public Address Book database. A user or server is given the access specified by the most specific entry that pertains to him – individual, group, or default. If several group entries all contain a user or server, then the maximum access level of all of those ACL entries is given. The documented exception to this is when a user is in two or more groups and is not mentioned explicitly in the ACL, and one of those groups is given NoAccess. In this case, the user will have no access, regardless of the permissions given to the other group.

The ACL also contains some information about how the ACL is to be maintained. The issue is whether each replica of a database will have a separately controlled ACL, or whether ACL changes will be propagated from one replica to another.

3.2.2.3 Some Additional Database Security Issues

Full Text Index of Databases

A full text index of a database is information that is used to support database searching. For subtle reasons, databases that have encryption-enabled fields should not have full text indices; otherwise, a security problem may arise.

Inheriting Future Design Changes

When basing a database on an existing "template" database, there are some potential security weaknesses:

- If the server of the template database has at least Designer access to this database (see section 3.2.2.2), and
- if this database inherits design changes from the template, and
- if a design update is triggered,

then anyone who can modify the design of the template can change the design of the database.

Databases Which Act As Templates

Any changes to the template database will propagate to those databases that inherit from it. Thus, if a user has a database that is intended for copying but not inheriting, and it can be converted to a template, then users who copy it will be given the confusing and dangerous choice of whether to inherit or not.

If a form is inherited from template databases, then the same concerns apply.

Controlling Database Size

It is possible to control the size of a database. Setting the maximum size of a database may prevent a denial of service attack in which flooding the database causes some *other* database to be unavailable.

3.2.3 Form Security

A form is a part of a database that describes basic characteristics upon which documents of the database can be built. The designer of a form can specify who can create documents with this form. This specification refines, but does not override, the permission that some identity (user or server) has to create documents in the database. This may be set at "All authors and above" or it may be an explicit list containing users, servers, groups, and roles.

The designer of a form can specify a default (initial) list of readers of any document created with the form. Again, this only refines database-level access. The creator of the document and any editors of it can adjust this reader access specification. The reader access can be either "All Readers and above" or an explicit list, as above.

Forms require Designer access (or higher) to edit them. Additionally, as forms (and other structural elements) are a kind of document, they are subject to the usual access controls for reading, editing, etc.

Forms may have a list of default encryption keys. When a document is created with this form, it will be encrypted with these keys, unless the author changes the list before saving. The person saving the document must posses the keys.

3.2.4 Document Security

A document may be specified as readable by "All Readers and above" or by an explicit list of readers consisting of users, servers, groups and roles. Upon creation, the initial setting is taken from the form, but authors and editors of the document can change it. We refer to this initial setting as the "document Readers ACL."

If the document contains a field whose type is "Readers," then any name listed there is also allowed to read the document. Such fields are highly configurable (by the designer) and may be editable in several ways, or may be computed. However, when used alone, controls on editing or computing Readers fields are weak security features, as they may be circumvented. When used in combination with other security features, a restricted Readers field may be considered part of the document security system. Also, since any editor of the document may add arbitrary entries to the document Readers ACL, a rigidly computed Readers field can serve only as a lower bound to who can read the document.

A document may contain a field whose type is Authors. If someone only has Author level access to a database, then that person can only edit (modify) documents that they have created. However, if there is an Authors field, then the names present there (users, groups, servers, roles) are allowed to edit the document, even if they only have Author access to the database. Note that, if there is an Authors field, the document creator has no special status — the creator must be listed if he is to have the right to edit his own document, once it is closed.

Like Readers fields, Authors fields may be editable or computed. If they are computed, it may be a security feature, depending on other related security features being used.

A document may be automatically encrypted when saved. The encryption keys that are used are either ones specified in the form or they are selected by the creator from those incorporated in the creator's ID. The effect of such encryption is to protect all encryption-enabled fields in the document. If a reader of an encrypted document does not have all of the keys, then the encrypted fields will appear blank. If the reader has all of the keys, then the encrypted fields will appear normally. Users can create such keys and can send them to one another. This kind of encryption is distinct from the encryption that happens on either end of an e-mail transaction.

Documents may be signed with the user's secret key either when they are mailed, or when they are saved. When a document is saved, if there are "sign-enabled" fields within access controlled sections of the document (see section 3.2.5), and if the user edited the document and was able to edit that section, then the enabled fields in the section will be signed, possibly overwriting any previous signature.

When a signed (saved) document is read, each access controlled section is labeled with the identity of the signature that is present, and the integrity of the section is checked. If the check cannot be performed, or if it fails, a warning is given.

3.2.5 Access Controlled Sections

An access controlled section of a document is a collapsible section that has two special functions for supporting signatures and edit access.

One purpose of an access controlled section is to specify that fields be signed when a document is saved. This is in contrast to a mailed document, in which all sign-enabled fields are signed by the sender. When a document with an access controlled section containing sign-enabled fields is viewed, the status of the signature is visible. Signing prevents tampering and is not repudiatable.

An access controlled section can also have a separate list of "Editors" which refines who (among those who can edit the document as a whole) can edit the fields in the section. This list of section editors can contain users, servers, groups, and roles. As a security feature, restricting editors is weak. A user who can edit a document can change the form of it (e.g., to a form where those fields are not controlled). The user may then edit those fields and change the form back. The only protection against this is that the section may also be signed. In this case, the altered document will have an invalid signature. The real use for edit-restricted sections is to allow users to edit documents containing signed fields, without invalidating the signatures.

3.2.6 Fields

Each field can have each of the following properties:

- encryption enabled,
- sign if mailed or saved in (access controlled) section, and
- must have at least editor access to use.

If a field is signed-enabled (as mentioned above), then, if the document is mailed, that field will be signed by the sender (if signing is used for that message).

When the document is saved, if the field occurs in an access controlled section and if the person saving has the potential to edit anything in that section, then the section will be signed by the saver. The signature will incorporate a hash of all sign-enabled fields in the section. This will wipe out any previous signature on that section.

When a document is read, the validity of any signatures is checked and the identity of the signer is displayed.

Users can incorporate additional secret (symmetric) keys into their certificates. They do this either by asking Lotus to generate a new key, or by receiving such a key from some other user (e.g., via e-mail) and "merging" the key into their certificate.

The fields of a document that are encryption-enabled may be encrypted with any set of such secret keys. When a user reads such a document, if she has all of those keys, the field will be readable. If she does not have all of the keys, then those encrypted fields will not be readable.

3.3 Further Information

As can be seen from section 3.2, Lotus Notes provides a complex mix of methods for controlling security. The purpose of the section was to provide sufficient information for the reader to understand the security policy language that we present below. For those desiring to use Lotus Notes, additional security information can be obtained from Lotus Notes manuals or a wide range of books about Lotus Notes.

4. Security Policy Language Overview

The specification language is designed to capture an application developer's access control intentions, as well as security characteristics of the application components. The language thus provides a way to specify and relate conceptual objectives with concrete design objectives.

The two objectives are primarily specified in the two layers of the specification language:

- a *Conceptual Layer* — for describing the data, users, and access rules
- a *Lotus Layer* — for expressing security characteristics of the Lotus components

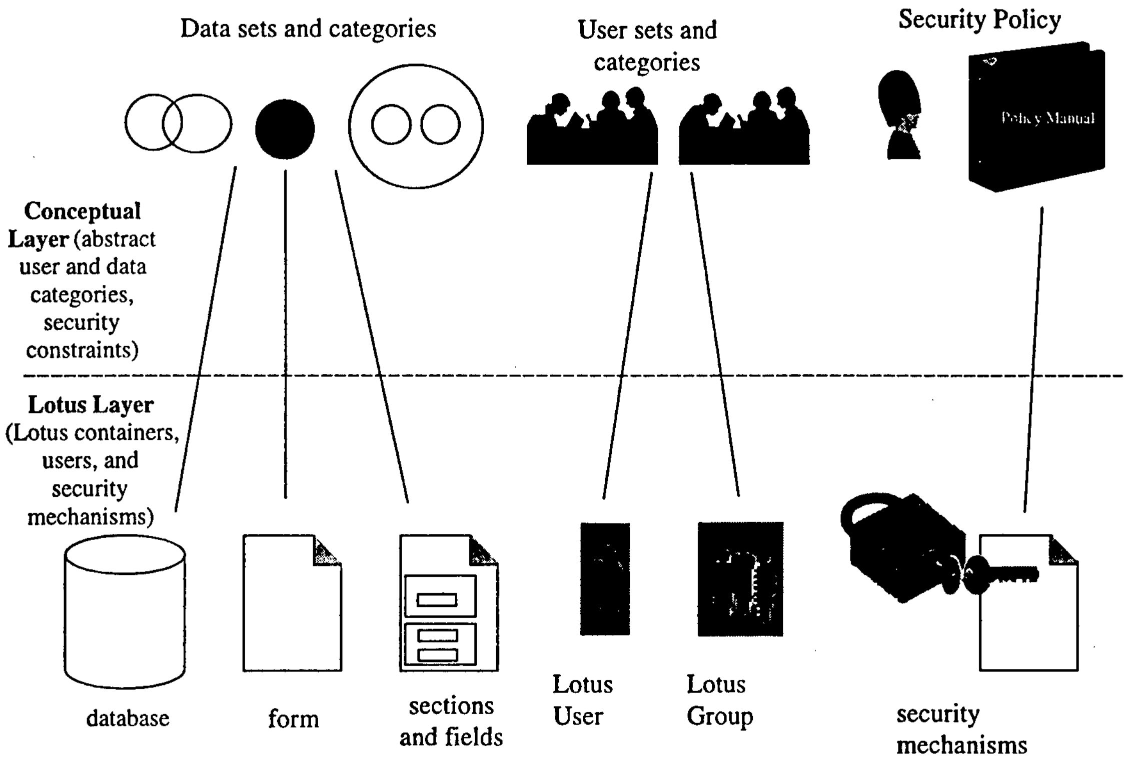

Figure 2: Conceptual and Lotus Layers

These layers are not completely independent. Relationships between the two layers are:

- what conceptual data is contained in a Lotus component,
- how are conceptual user classes mapped to Lotus users and groups, and
- how are the conceptual strengths needed for access rules related to the actual enforcement mechanism (which we call *seriousness* of enforcement)

4.1 Conceptual Layer

The specification language supports an organized way of grouping users and data. These groupings could be done arbitrarily. However, the intention is that they be used to identify classes of users and data that should be treated similarly by the Lotus Notes security controls.

The conceptual layer also provides an abstract explanation of the desired access rules. It supports the expression of positive access rules associated with what access is needed to accomplish an activity, as well as negative access rules that are needed to enforce the desired security policy constraints.

Specifying access limitations (negative rules) is relatively straightforward, but it may involve combining many security concerns. The policy language provides a way of expressing each concern separately.

Specifying how the data should be controlled to provide sufficient access is more difficult. This is because the security protections need to be compatible with functional constraints. In the policy language, positive access requirements are grouped together in an "activity."

One also needs to characterize the kind of assurance needed for various kinds of access restrictions. Just specifying that a kind of data needs to be hidden from some set of users does not provide information on how well protected it needs to be. A seriousness identifier may be associated with an access rule. Each such identifier should have an informal conceptual explanation of how important it is to enforce that rule. This identifier should also be associated with informal explanations about possible implementation methods at the Lotus layer.

4.2 Lotus Layer

To evaluate whether a conceptual policy will be properly enforced, there must be details about the intended Lotus layer security controls. It is particularly important to carefully specify details about the controls for Lotus Notes, because there is such a wide range of controls.

The Lotus layer specification consists of a description of
- how various components are organized, e.g., what are the fields of a document, and
- security attributes for each of these pieces, e.g., whether a database is encrypted

Different kinds of components have different kinds of attributes and this is reflected in the grammar of the language.

This layer of the language also provides syntax for relating Lotus objects with the conceptual data that they can contain. Additionally, an informal description of the methods applicable for enforcing different importance levels (seriousness) for the access rules can be expressed.

4.3 Syntax Conventions for Describing the Language

The grammar is described using an extended BNF:

- Keywords are boldfaced and one character keywords are double quoted.
- Options for expanding a grammatical category are separated by a vertical bar.
- Items can be grouped together with parentheses.
- A star following an expression indicates zero or more occurrences.
- A plus following an expression indicates one or more occurrences.
- Optional items are enclosed in square brackets.

- A comma-separated list of items is indicated by prefixing the kinds of items in the list with the word *listof.*

Comments in the grammar begin with a #.

Identifiers in the language are indicated by `<Identifier>`. Identifiers are case sensitive. They may include either underscores or dashes. Identifiers may also be introduced using a more descriptive category name that expands to `<Identifier>`. For example, the category `<principal>` maps to an identifier that represents a user, server, or group. The grammar rule for this is `<principal>=<Identifier>`. An identifier must typically be declared before it is used. Some statements simultaneous declare and constrain items represented by identifiers.

A string is any text enclosed in double-quotes. Double-quotes inserted in a string must be prefixed with a backslash.

A specification may contain comments on any line. They are prefixed with a # and run to the end of the line.

The starting non-terminal for the grammar is `<specification>`. A specification consists of a conceptual and a Lotus layer specification

```
<specification> = <conceptual_specification> <lotus_specification>
```

The two parts of the language are described in sections 5 and 6.

4.4 Some Aspects of the Language

4.4.1 Formal and Informal Parts of the Policy Language.

The expression of the policy and its design consists of both a formal specification (that conforms to some grammar) as well as informal annotations. Some of the annotations, such as seriousness of enforcement, are separate parts of the expression of the policy. Other informal annotations that are closely associated with particular objects, or attributes, are included as strings within the grammar. Additionally, clarifying comments will normally be a part of a specification.

4.4.2 Constructing a Specification and the Language Syntax

We envision some kind of user interface to help formulate a specification. This is described briefly in section 8.1. In this situation, the particular language syntax is not very important, as the user need not directly use the language. However, to clarify this presentation and to provide a possible exportable version of a specification, a syntax is presented along with the semantics.

4.5 What is not Included in the Language

4.5.1 Temporal Aspects of Policy

In addition to describing the static characteristics of how data should be protected, it is often useful to describe temporal restrictions. Two kinds of policies for temporal characteristics are:

- Workflow authorizations and changes in access policy
- Adaptive policies

Specific data changes that may be worth modeling are:

- a change in the security attributes of some data,
- creation of new data,
- deletion and invalidation of data, and
- generation of protection attributes associated with data such as signatures.

Controlling this kind of security using just Lotus Notes is complex. We describe some possible extensions to Lotus Notes 4.1 that might make such controls more useable in section 10.3.

4.5.2 Navigators and Views

The design of navigators and views has few security concerns, so the policy language does not cover this.

4.5.3 Specialized Language Features for Common Paradigms

A general language for describing a policy may be difficult to use when trying to capture specialized concerns for specific application domains. One possibility is to have specialized policy languages that are appropriate for particular classes of examples. By picking some particular cases for common paradigms, the language will be more closely connected with the actual activity and easier to use properly.

An alternative is to have just a general language and provide a library of examples illustrating common strategies. In this case, no language extension would be needed.

This report does not cover this topic.

5. Conceptual Layer of the Language

5.1 Top Level

The conceptual specification is a grouping of different types of statements.

```
<conceptual_spec> = Conceptual <conceptual_statements> End Conceptual
```

```
<conceptual_statements> = <conceptual_statement> ";"
                  | <conceputal_statement> ";" <conceptual_statements>

<conceputal_statement> = <declaration> | <user_data_statement> |
             <function_declaration>
             | <activity_statement> | <factor_statement>
```

The categories <declarations> and <user_data_statements> describe classes of uses and data. The category <function_declaration> introduces functions that relate individual items to classes for use in access rules. Access rules are described as part of an <activity_statement> or <factor_statement>.

5.2 Data and Users

Two basic types of items can be expressed in the conceptual layer:

- sets of users
- sets of data

This part of the language establishes distinctions among users, or data, so that the distinct classes may be treated differently in the implementation. A specification says what the classes are and how the classes must be treated for access control.

Determining, or expressing, the class in which a datum or user should be contained is frequently a matter of judgement and is difficult to capture formally. For example, an organization may establish rules for handling *Proprietary* data, but have only guidelines for what data should be considered *Proprietary*. Thus, the formal part of the specification should be augmented by the necessary informal justifications of why certain kinds of data are in certain classes.

5.2.1 Language Definition

Data and users can just be declared as classes, or they can be given full or partial definitions.

A declaration introduces the class names. Class names are global for the entire specification, not just the conceptual part of the language.

```
<declaration> = (User | Users) <Identifier>
              | (Data | Datum) <Identifier>
```

Note that keywords **User** and **Datum** may appear in either the singular or the plural form. The singular form indicates a singleton set. (The singular form may also be thought of as indicating a particular individual element, with appropriate overloading of operations that mention it.)

More information about a class may be given by a `<user_data_statement>` statement. This statement also globally declares the class. Note that, syntactically, more then one `<user_data_statement> or <declaration>` is permitted for a class, although this may introduce an inconsistency in a specification.

```
<user_data_statement> = <declaration> (<qualifier_expression>)+
```

Several kinds of qualifier expressions can be used to form the user and data classes.

```
<qualifier_expression> = <defn_qual> | <subset_qual> |
                  <superset_qual> | <partition_qual> |
                  <disjoint_qual> | <attribute_qual>
```

The `<defn_qual>` statement is used to express definitions from standard set theory: union, intersection, and complement. For convenience, there is also a keyword for set difference. Union, Intersection, and Difference may be used either as infix operations or as prefix operations applied to a set of items.

```
<defn_qual> = defined as <expression>
            | equals <expression>

<expression> = <Identifier> | ALL_USERS | ALL_DATA
                  | <expression> set_op <expression>
                  |  <set_op> "{" listof <expression> "}"
                  | Complement <expression>
                  | ( <expression> )

<set_op> = Union | Intersection | Difference
```

Unions and intersections of sets must have arguments of the same type (*User* or *Data*). The complement of a set is with respect to the set of all elements of the same type.

A partial specification using the subset relationship is also supported.

```
<subset_qual> = subset <expression>
                  | in <expression>    #useful for singletons

<superset_qual> = contains <expression>
```

Two additional methods for describing the elements of a class are the expression of partitions, which are disjoints unions, and the assertion that two sets are disjoint. The same semantics can be expressed with the more basic syntax, described above, but this syntax is more concise.

```
<partition_qual> = partitioned by
              "{" <expression>
                  ( "," <expression> )* "}" )   [expresses <Identifier>]

<disjoint_qual> = disjoint from <expression>
```

If any of the expressions in the list for a partition statement is an identifier, then this statement also serves to declare a new class.

Attributes

Note that the partition function has an optional "**expresses**" identifier, which associates an "attribute" with a partition rule. The identifier is intended to name some characteristic that motivates the intention of the partition. An informal description should usually be associated with an attribute identifier.

One type of attribute that may be useful is the stage in the lifecycle of some data. This could be represented as a Finite State Machine (FSM) attached to each datum, which records it life stage. We recommend a simplified view of lifecycle: where similar states of the FSM are grouped together as stages. The *stages* attribute associated with a partition is thus a "hint" (not supported by any semantics) that data may make transitions from one element of the partition to another, depending on its stage in the lifecycle.

5.2.2 Intended Usage

Sets of users and sets of data are formed in order to support the description of security policies in which some users (or data) are treated differently than others. The set definitions are the abstractions that correspond to distinctions (among users or among data) that one intends to draw in the application. For instance, one may wish to treat classified data differently from unclassified data. However, if one intends to treat secret and confidential data similarly, there is no need to mention the set of secret data or the set of confidential data. This form of parsimony (refraining from creating spurious sets) does not prevent us from making extra distinctions for specification clarity or for distinguishing other aspects of security that are not part of the model.

Here are some examples of distinctions among *data* that might be relevant to a security policy:

- Hierarchical importance levels and categories (Even for non-military uses, some form of hierarchical classification of the importance of protecting the data is useful.)
- Application area specific need-to-know and need-to-modify grouping.
- Integrity properties.
- Changes in access required as the data is processed.

Here are some examples of distinctions among *users* that might be relevant to a security policy:

- User clearances.
- Groups.
- Job/Responsibilities.
- Capabilities and Qualifications for possible future change in rights.
- Delegation rights.
- Special rights, such as those involved in certification or administration.

5.2.3 Example

```
Users Odyssey;
Users Admin;
Users Technical;
Users Trusted;
Users Vanilla;
Users Odyssey partitioned by {Admin, Technical};
Users Odyssey partitioned by {Trusted, Vanilla};
Users Comptetitors disjoint from Odyssey;
Users StarMtn contains Odyssey disjoint from Competitors;
User GenMgr in Odyssey;
Users ProjMgrs defined as Intersection(Technical, Trusted)
               contains GenMgr;

Data Odyssey_data;
Data Financial_data in Odyssey_data;
Data Personel_records;
Data Salaries;
Data Employment_history;
Data Financial_data contains Salaries;
Data Executive_salaries in Salaries;
Datum Salary_database in Financial_data;

Data Personel_records partitioned by {Salaries, Employment_history}
   expresses personel_record_sensitivity;
```

5.3 Functions

Function declaration are used to help express access rules. For instance, one may want to express the special rights that each user has to the data that he owns.

```
<function_declaration> = Function <Identifier> from <expression> to
                                                      <expression>
```

These functions map an element from a data or user class to another subset of data or users. The subsets in the range may be singletons, but this is not a requirement. (The range is technically the "powerset" of some data or user class.)

Function declarations are also useful in establishing a connection between the conceptual and concrete layers (see section 7). Hence, these declarations are global.

Note that no syntax is provided for partial or full definitions of functions. Informal annotations should be added where appropriate. The annotation can be a complete description of the function, or simply some aspect of the function. For example, it may be useful to indicate that a function is partial, into, onto, or one-to-one.

5.4 Access Specification

Access rules are used for specifying that certain accesses are needed (positive rules) and for specifying certain accesses are prohibited (negative rules). These are grouped as *activities* and *limitation factors* respectively.

5.4.1 Activities

Activities are groupings of access control rules that express the required access needed to perform some activity. Examples of activities might be financial record keeping, payroll, software development, project management, or proposal writing.

For clarity, user and data declarations may appear within an activity to suggest that they are only appropriate for that activity, but they are still global declarations.

```
<activity_statement> = Activity <Identifier>
    [declarations
      (<user_data_statement> )+ ]
     required <access_rules> ";"

<access_rules> = <access_rule> ( ";" <access_rule> )*
```

The operations that may be named in access requirements are determined by the functionality of Lotus Notes:

- **Read**
- **Create**
- **Edit**
- **Delete**
- **Sign**: can affix a signature to the data.
- **Design**: can alter the "design" of the data, i.e. edit the form
- **Manage**: can control and alter access to the data.

These capabilities are not strictly orthogonal.

```
<LotusVerbs>= <LotusVerb> | "(" <LotusVerb> ("," <LotusVerb> )+ ")"

<LotusVerb> = Read | Edit | Author | Create | Delete | Sign
               | Design | Manage
```

There are two forms of access rules.

```
<access_rule> = "{" (<Identifier> ","
                      <Identifier> ","
                      <LotusVerbs> "}"  [seriousness <Identifier>]
                 | <Identifier> ":" <Identifier>
                           "{" <Identifier> ","
                               <Identifier> "(" <Identifier> ")" ","
                               <LotusVerbs>  "}"
                            [seriousness <Identifier>]
```

The first is a similar to a classical access triple:

```
{U, D, Access_Verbs}.
```

U is a class of users, *D* is a class of data and *Access_Verbs* is a list of verbs taken from a fixed set. The intended meaning is that all users in *U* must be able to perform any of the listed verbs on any datum in *D*.

The second is a parameterized form, which can be used in two ways. The first way expresses access requirements for each individual user in a class to data that is functionally related to him.

```
x : U {x, f(x), Access_Verbs}
```

The function *f* must be a declared function with a domain that is a set of users, *U*, and a range that is a set of data. The intended meaning is that each user, *x* in *U*, must be able to perform any of the stated verbs on any datum in *f(x)*. For example, one could require that each user have certain access to *their own* documents. In this case, a function that maps each user to the class of data that is "theirs" would be declared, and referred to in the requirement.

The second parameterized form is similar to the first, except that it uses functions from data to users.

```
y : D {g(y), y, Access_Verbs}
```

g must be a declared function with domain a data set, *D*.

Associated with each access rule may be some indication of how serious, or important, it is to enforce that rule. This is often needed since there are multiple ways that access can be enforced, and they do not have the same security characteristics. The language supports using a named **seriousness** indication. Informal descriptions of the seriousness need to be provided. Also, as part of the Lotus layer, some indication needs to be given of possible implementation choices.

Some possible implementation choices that could correspond to named degrees of seriousness (of access rules) include:

- by policy, but not enforced
- by obscurity (only weakly enforced), for example, using hidden fields
- by access control lists
- by signed hash (for integrity checking)
- by encryption with limited key distribution

Example of Activity

```
Activity payroll
  declarations
    User paymaster defined as Admin Intersection Trusted ;
    User source   defined as Salary_database Union timesheets;
  required
     { paymaster, source, Read } seriousness critical;
     { paymaster, register, (Read, Edit) };
     { GenMgr, CheckRegister, Read };
     { GenMgr, Checkbook, (Create, Read)} ;
```

5.4.2 Limitation Factors

Limitation factors are restrictions on access. Each factor is an interlocked set of restrictions, which may spring from a common need, and which may have an implementation strategy in common. The overall system is intended to be subject to the *conjunction* of all of the factors. Factors are intended to be independent of one another.

Each factor consists of a set of individual *restrictions* and possibly some additional declarations. Note that any declaration appearing in a factor is still global.

The access rules have the same syntax as for activities. In particular, restrictions take the same two forms as access requirements. However, they are interpreted as *forbidding* the stated access.

```
<factor_statement>  = Factor <Identifier>
 [ declarations
     (<user_data_statement> )+ ]
   denies <access_rules> ";"
```

Example of Factor

```
Factor privacy_of_salaries
  denies {Trusted_admin, salary_database, Read}
         {GenMger, Salaries, Edit};
```

6. Lotus Layer

The Lotus layer of the specification describes the hierarchical structure of the Lotus objects and security attributes associated with those objects.

6.1 General Remarks

6.1.1 Lotus Components

A specification in this part of the language describes one or more *organizations*. Each organization *contains* some users, servers, and organizational units. Each of those, in turn, contains other components. We use the term *component* to describe any organization, organizational unit, server, user, or any similar smaller structure in a specification. Each component is, in some sense, a unit of implementation for Lotus Notes applications. When one component is contained by another, it is called a *subcomponent.* So, the specification describes the hierarchy of components that make up the organization.

When we speak of the *kind* of a component, we are referring to its primary attribute,e.g., *database* or *server.* Each kind of component may have only certain subcomponents. as determined by the architecture of Lotus Notes. For example, a database may not contain a server.

The specification language permits a partial specification of subcomponents. For example, one may specify only some of the databases on a server. One may also specify that specified subcomponents of a particular kind are exhaustive. The "`Exhaustive for`" keywords are used to express this indication. For instance, in a Server one might specify "`Exhaustive for Databases`" to indicate that this server contains only, and exactly, the databases listed.

All component declarations have a similar syntactic structure, that is, they begin with a reserved word that names the kind of component, followed by an identifier that names the component within the specification. Component declarations end with a reserved word constructed with the prefix "End." In between are expressions that characterize the component. This includes any subcomponents and any "settings" that may be relevant to security. The particular kinds of subcomponents and the particular kinds of attributes that are allowed depend on the kind of component.

6.1.2 Component Attribute Specification

Each component may have some attributes declared in the specification. As with subcomponents, the possible attributes (and their values) for any component are determined by the component kind. An attribute that is omitted is considered to be unspecified. Note that this is different from giving it a default value.

Some of the attributes take their values from structured types and other attributes have values that are strings. Strings are used when there is a wide range of characteristics to be captured and no standard representation of this information. If more structuring is desired, the grammar for particular attributes could be extended.

6.1.3 Patterns

Some components are designated as "patterns." This indicates that it describes not an individual component, but rather a class of similar structures. These are called the *instances* of the pattern. If nothing further is said, the pattern refers either to instances that are dynamically created and destroyed, or to some unknown but fixed setof instances.

In the syntax, if the token following a component name is **pattern**, then the specified structure is a pattern.

There are two ways to relate a pattern to its instances.

```
<instanceDecls> =                 #list of instances for a pattern
   has instances <identifierList> [is exhaustive]

<instanceOfDecl> =
   instance of <pattern_name>   #the pattern for a particular component
```

(See section 6.1.6 for the context of this syntax.)

6.1.3.1 Parameters

Different instances of a pattern may have different components associated with those instances. For example, different databases may have different owners. Parameters are used to name those components. Parameter values can be restricted to fall under some component called its ancestor.

A parameter declaration contains the name of the parameter, its kind (e.g., User or Database), possibly the name of a component or pattern that is the ancestor, and possibly a function associated with the parameter.

Parameters can be thought of as corresponding to functions as described in section 5.3. For example, in a database pattern, `D`, the parameter `owner` of kind `User` can be thought of as a function which when given a particular instance of `D` returns the user that is the owner. The domain of the function is the class of items described by the pattern. The range of the function is the class of items described by the parameter kind. The function part of a parameter declaration is for establishing the connection to the conceptual layer function.

The syntax for declaring parameters is:

```
<parameter> = parameter <paramName>
                   of kind <Kind>
                  [ in component <component_name> ]
                  [ is function <Identifier>]
```

(See section 6.1.6 for the context of this syntax.)

6.1.3.2 Name Resolution

When constructing attributes of a component, one may want to refer to some other component or instance. For example, a group may be part of a database ACL. To do this, one uses the name of the component (in this case, the group name). Since these names have global scope, there is no ambiguity in what they reference. However, the interpretation of what this reference means for patterns requires more explanation. There are three distinct possibilities:

- All of the instances of the pattern refer to the same component.
- Each instance of the pattern refers to a distinct component.
- Unspecified – each instance refers to a component that may be distinct.

Typically, the semantics of this language supports the first of these. When referring by name to a component from within a pattern, we mean that all instances of the pattern refer to the same thing. The first exception to this rule is when a component names another component (or instance), and both are part of the same pattern, then the second interpretation applies. The other exception is for names that are parameters. The parameter in different instances of a pattern may, or may not, refer to the same thing — it is unspecified.

6.1.3.3 Controlling Pattern Instantiation

Components may be declared to be instances of a given pattern. Both the pattern characteristics and specific component characteristics apply. The pattern and component should be siblings.

Care must be taken to avoid inconsistencies. Any attributes or subcomponents that are declared should not conflict. Also, the *presence* of an instance must be consistent with the instance declaration of the pattern, if any.

When one pattern is declared to be an instance of another pattern, this is interpreted as a specialization of the pattern. The same restrictions on consistency apply.

Patterns interact with exhaustiveness declarations as follows. If a pattern is included in an exhaustive list of subcomponents, then the true complete list of subcomponents includes any instances of that pattern, as described above.

6.1.4 Users

In the explanation of the Lotus layer specification, we frequently designate users or principals as the values of attributes. We do *not* mean that a specification should list actual members of the organization by name. Rather, these should be the names of other components of the specification whose *kind* is User. Such entities are intended to be placeholders in the specification for an actual person in the organization.

6.1.5 Contains and Implements

`contains` and `implements` declarations provide a way of connecting the Lotus layer and conceptual layer specifications. These are described in section 7.

6.1.6 General Component Grammar

This section contains a general representation of the syntax of all components. Section 6.2 has the specific grammar rules for each type of component.

Each of the component specifications has a similar grammatical structure. The schema for grammar productions (rules) for an arbitrary *kind* of component (where "kind" is a grammar meta-variable, to be replaced by the actual kind of the component) is:

```
<Kind Component> = Kind  <Identifier> [pattern]
     [<instanceDecls>]
     [<parameterDecls>]
     [<instanceOfDecl>]
     [<containsDecl>]
     [<implementsDecl>]
     [<attributes>]
     [<subcomponentSpec>]
EndKind

<instanceDecls> = has instances <identifierList> [is exhaustive]

<parameterDecls> = <parameter> | <parameter> <parameterDecls>

<parameter> = parameter <paramName>
                        of kind <Kind>
                       [ in component <component_name> ]

<Kind> = Organization | OrgUnit |Server | Workstation | Database
              | Form | Section | Field | Document |User | Group

<Kind_plural> = Organizations | OrgUnits |Servers | Workstations |
        Databases | Forms | Sections | Fields | Documents |
        Users | Groups

<instanceOfDecl> = instance of <pattern_name>

<subcomponentSpec> = [<subcomponentList>] [<exhaustivenessClauses>]

<subcomponentList> = <component(appropriate kind)> |
                     <component(appropriate kind)> <subcomponentList>

<exhaustivenessClauses> = Exhaustive for <Kind_plural> |
                     Exhaustive for <Kind_plural> <exhaustivenessClauses>

<containsDecl> = contains  (listof <Identifier)
               | contains only  (listof <Identifier>)
<implementsDecl> = implements <Identifier>
```

In each of the following sections, the grammar productions (rules) for a particular kind of component are given. It should be understood that the general structure, given above, holds for all of them and will not be repeated *in detail.* Those sections concentrate on the syntax of attributes that are specific to the kind of component and the allowed kinds of subcomponents to the kind of component.

We introduce the following identifier categories and list cateogries.

```
<principal>=<Identifier> #user, server, or group.
<userName>=<Identifier>
<component_name>=<Identifier>
<pattern_name>=<Identifier>
<paramName>=<Identifier>

<userList> = listof <userName>
<principalList> = listof <principal>
<keyList> = listof <key>   #list of encryption keys
```

6.2 Component and Item Descriptions

This section describes all of the *kinds* of components. The description of each kind of component contains the following:

- A description of the component and a fragment of the grammar that shows the allowable attributes and subcomponents.
- A list of the intended meanings and allowed range of attribute values.
- A list of the allowed subcomponents.
- An example.

Lotus specifications could start at a variety of levels in the hierarchy. However, specifications need to start high enough to capture essential security information, even if this information is common among many applications. For this reason,

```
<lotus_specification> = <organization> | <org_unit>
```

6.2.1 Organization

An organization describes the top-level certificate authority for a Lotus Notes installation.

```
<organization> = Organization <Identifier> [Pattern]
                    [Owner <userName>]
                    [DomainManager <principalList>]
                    ( <orgunit> |
                      <server> |
                      <user> |
                      <group> |
                      <workstation> )*
                 EndOrganization
```

The specification begins by declaring an organization (or pattern for an organization).

At present, we support only specifications with a single top-level certification authority, and thus, a single domain.

Large organizations may use multiple Lotus Notes domains. Interoperability between domains requires cross-certification and possibly explicit assignment of trust. Such systems are currently beyond the scope of this specification language.

Attributes:

Owner. This is the person who controls the certification identity. Lotus recommends that the certification identity be written on a disk and stored off-line in a safe place. Thus, the owner should be the person who has control of this disk.

Domain manager. This is the list of people or groups who have the right to alter and maintain the Public Address Book. We assume that this list is the same for all replicas of the Public Address Book. We also assume that all of these users can perform all maintenance operations. Not listed here are the server identities with administrative access, since that may differ from one replica to another.

Composed of:
Organizational Units, Servers, Users, Groups, and Workstations.

Example Fragment:

```
Organization Odyssey
   Owner security_officer
   Manager admin_group
   User security_officer etc EndUser
   Group admin_group etc EndGroup
     other components
EndOrganization
```

6.2.2 Organizational Unit

The organizational unit is a secondary certificate authority. Its identity is certified by either the organization certifier, or another organizational unit certifier that is "closer" to the organization.

```
<orgunit> = OrgUnit <Identifier> [Pattern]
                     [Owner <userName>]
                     ( <orgunit> |
                       <server> |
                       <user> |
                       <group> )*
                  EndOrgUnit
```

Attributes:
Owner. This is the person who controls the certification identity

Composed of:
Organizational Units, Servers, Users, and Groups

Example Fragment:

```
OrgUnit Odyssey_Ithaca
  Owner security_officer_ithaca
  User security_officer_ithaca etc EndUser
     other components
EndOrgUnit
```

6.2.3 Server

A Notes server is designated by a certified identity. It has an installed base of software on some server machine and has a replica of the Public Address Book.

```
<server > = Server <Identifier> [Pattern]
                      [PAB        [(master | slave)]
                                  [(encrypted | plain)]
                                  [(protected | unprotected)]
                                  [(compareKeys| dontCompareKeys)]
                                  [(PABbasedAccess| DefaultAccess)]   ]
                      [ Manager <userList> ]
                      [ Machine <String>]
                      [ PlatformSecurity <userList>   ]
                      [ OSLogin (restricted_local <userList> |
                                 local                       |
                                 restricted       <userList> |
                                 strong                      |
                                 weak_or_none)                 ]
                      [ OSFileProtection <boolean>                 ]
                      [ AllowAccess (<principalListOrAnybody>    ]
                      [ DenyAccess   <principalListOrAnybody>    ]
                      [ CreateDatabases <principalListOrAnybody>]
                      [ CreateReplicas  <principalListOrAnybody>]
                      [ RunPersonalAgents <principalListOrAnybody> ]
                      [ RunRestrictedLSAgents <principalListOrAnybody>]
                      [ RunUnrestrictedLSAgents
                                  <principalListOrAnybody> ]
                         <database> )*
                   EndServer
```

<principalListOrAnybody> = <principalList> | **nobody** | **anybody**

Attributes:

Public Address Book (PAB):

master or slave. This describes whether changes to this server's replica of the Public Address Book will be propagated to other replicas or not. Notice that this does not capture the most general relationship possible between database replicas. However, only these two cases are really useful in the case of the Public Address Book.

encrypted or not. Describes whether the replica of the Public Address Book for this server is encrypted by the server's public key.

protected or not. Describes whether the database file is directly accessible through the OS (to a wide range of users).

compare keys or not. Describes whether, as part of the authentication process, the public key presented by a (potential) user will be compared with the value stored in the PAB.

PAB access or not. Describes whether the PAB will be used as a kind of ACL for this server.

Manager: list of users and groups. Those who are listed as managers of this server in the Public Address Book.

Machine: name and location. Each of these is optional and has no real impact on security. However, they are of practical importance, so they can be recorded in the specification.

Platform Security: list of users who have physical access to the machine, if this is, indeed, a restricted list.

OS login: one of

- restricted local. Login is only allowed from the console. The list of authorized users is small, and is given.
- local. Login is allowed only from the console.
- restricted.. The list of authorized users is small, and is given.
- strong. A reasonably strong authentication method is used to grant access to the machine, either locally or remotely. No access is allowed without authentication and authorization, though the list of authorized users may be large.
- weak or none. Either authentication for login is based on a weak method or is not done at all, or access is granted even without successful login.

OS File Protection: boolean. Denotes whether the default for database and identity files on this server is that they will have suitable OS access control protection.

Allow Access: list of users, groups, servers. Only the listed principals should be allowed access to this server.

Deny Access: list of users, groups, servers. The listed principals should not have access to this server. This takes precedence over the Allow Access, if there is a conflict with regard to some specific user.

Create Databases: list of users, groups, servers. Principals who are allowed to create new databases on this server

Create Replicas: list of users, groups, servers. Principals who are allowed to create replica databases on this server.

Run Personal Agents: list of users, groups, and servers. Those who can run personal agents on this server. There is no default.

Run Restricted LotusScript Agents: list of users, groups, and servers. Those who can run restricted LotusScript agents on this server. There is no default.

Run Unrestricted LotusScript Agents: list of users, groups, and servers. Those who can run Unrestricted LotusScript agents on this server. There is no default.

Composed of:
Databases

Example Fragment:

```
Server TravelServer
   PAB slave encrypted dontCompareKeys DefaultAccess unprotected
   Manager  travel_director, admin_group
   Machine "The NT machine in room A22"
   PlatformSecurity travel_director, janitor
   OSLogin  local
   OSFileProtection False
   DenyAccess        TerminatedEmployees_Group, Janitor
   CreateDatabases   nobody
   CreateReplicas    nobody
   RunPersonalAgents nobody
   RunRestrictedLSAgents anybody
   RunUnrestrictedLSAgents  travel_director
EndServer
```

6.2.4 Workstation

This is intended to describe a user's workstation. Typically, this aspect of Notes can be ignored, unless replicated copies of databases will be stored on the workstation. In that case, the issue will be whether they are encrypted (for the user) or protected in some other way.

```
<workstation> = Workstation <Identifier> [Pattern]
                    [ Location <String> ]
                    [ PlatformSecurity <userList>  ]
                    [ OSLogin (restricted_local <userList> |
                              local                        |
                              restricted       <userList> |
                              strong                       |
                              weak_or_none)                   ]
                    [ OSFileProtection <boolean> ]
                       (<database> )*
                  EndWorkstation
```

Attributes:

Location: physical location of the server.

Platform Security: (see Server)

OS login: (see Server)

OS File Protection: (see Server)

Composed of:
Databases

Example Fragment:

This example fragment is a bit more complicated than the earlier ones: it is a pattern and it uses a parameter. The user of a workstation that follows this pattern is given physical access to the workstation, but there is no strong login mechanism. The user is allowed to have only one local database, a replica of the rolodex.

```
Workstation typical_accountant_ws pattern
  parameter thisAccountant of kind User in component accountants
  Location "Accoutning department offices"
  PlatformSecurity thisAccountant, AccountingDeptHead, sysadmin_group
  OSLogin weak_or_none
  OSFileProtection False

  Database rolodex_replica etc EndDatabase
  Exhaustive for database
EndWorkstation
```

6.2.5 Databases

A database is a collection of data objects with design information. Note that certain aspects of databases are not relevant to our analysis (for instance, Views and Folders) and they are not described in the language. (Although, the ability to create new private views or folders is a right which may be restricted.)

```
<database> = Database <Identifier> [Pattern]
   [encrypted locally]
   [Roles (<role>)*]
   [ACL [prefix] <AclEntry>*]
   [Replica of <database_name> [<string>] ]
   [InheritsFromTemplate <database> <string>]
                                #(What aspect of design or access does it
                                #inherit?)
      (<form>|<document>)*
   EndDatabase


<ACLEntry> = "{" (<principal> | Default ) "," <level_of_access>
                 ["," With (<optional_access>)* ]
                 [ "," Roles( <role>+ | none ) ] "}"

<level_of_access> = manager | designer | editor |
                    author | reader | depositor| noAccess
<optional_access> = ( none | create | delete | createPersonalAgents |
                    createSharedFolders | lotusscriptAgents )*
```

Attributes:

Local encryption: boolean. Indicates whether the database is locally encrypted.

Roles: list of identifiers. The roles that may be assigned to any principal mentioned in the ACL. Some description of the purpose of the roles should be informally specified.

ACL: list of ACL entries. Each entry consists of a principal, level of access, and possibly optional access capabilities or list of roles. Also, the value contains an indication if this list of entries is a "prefix." If the indication is not present, then it is assumed to be the complete ACL.

Replicas: string. Describes the databases that are replicas of this one, and their relationship to one another (e.g., master/slave).

Templates: string. Describes the security-relevant features inherited from a template, if any. See section 3.2.2.3 for a description of the issue to be addressed.

Indices: boolean. Whether to allow full text indices for this database. See section 3.2.2.3 for a description of the issue to be addressed.

Composed of:
Forms, Documents

Example Fragment:

```
Database conference_planning
  encrypted locally
  Roles conferenceCommitteeChair
  ACL prefix {Default,depositor}
             {conferenceCommittee,editor,
                  With delete createSharedFolders}
             {sys_admin_group,manager}
EndDatabase
```

6.2.6 Forms

Forms define the structure of documents. They contain fields and sections (which may also contain fields). This includes some special fields that affect security. Forms also have some security-relevant "properties" such as the default readers list.

When a document is created with a form, the document is (in the terminology of this report) a subcomponent of the form. By contrast, an instance of a form pattern is a form or form pattern.

Note: A form is very similar to a document pattern, since both describe collections of similar documents. However, they are not exactly equivalent. Typically, a form should be used when the intent is that a form will be implemented in Lotus Notes.

Note that the documents of a form may be mentioned under the form or directly under the database.

```
<form> = Form <Identifier> [Pattern]
            [Creators (<principalList>|All_readers_and_above)]
            [DefaultReaders (<principalList>|All_authors_and_above)]
            [Readers <string>]
            [Authors <string>]
            [Keys  <keyList>]
             ( <section> |
               <field>   |
               <document> )*
EndForm
```

Attributes:

Creators: list of principals or **All_authors_and_above.** Those who can create documents with this form.

Default readers: list of principals or **All_readers_and_above.** This is the default read access list of newly created documents based on this form. The creator of the document can alter this list, if they are using the Notes user interface.

Readers: string. This indicates a field in the form of type "Readers" is present. The string should indicate whether the value is initialized, computed, or editable.

Authors: string. This indicates a field in the form of type "Authors" is present. The string should indicate whether the value is initialized, computed, or editable.

Keys: list of identifiers. This is the *default* list of keys that should be used to encrypt all documents created with this form. Users can override this list by changing the keys property of a document before saving.

Composed of:
Sections, Fields, Documents

Example Fragment:

```
Form schedule_form
  Creators admin_group
  ReadersDefault  employees_group
EndForm
```

6.2.7 Section

A section is a part of a document, so it can appear under Form, Document, and Section. Each section contains fields, has a title, may be hidden or visible, may be collapsed, and so on. AccessControlled sections are ones that may enable signing of some of their fields. They may also further restrict who can edit them.

Only access controlled sections are relevant for security. Thus, section declarations appearing in the formal specification refer to access controlled sections. (Section declarations that are not access controlled can be described with comments.)

```
<section> = Section <Identifier> [Pattern]
           [Editors (<principalList> | <string>) ]
           [Signed <userName>]
            ( <section> |
              <field>    )*
          EndSection
```

Attributes:

Editors: list of principals or string. If this section is in a document, then the list of principals are those who can edit the section. If the section is in a form, then the string describes whether the field is computed and, if so, how.

Signed: user. This attribute specifies the name of the signer of the section. It can be useful in specifying sections of documents or document patterns.

Composed of:
Sections, Fields

Example Fragment:

```
Section sender_signature
   Field etc EndField
EndSection
```

6.2.8 Fields

```
<field> = Field <Identifier> [Pattern]
            [Encrypt_Enable  <boolean>]
            [Sign_Enable     <boolean>]
          EndField
```

Attributes:

Encryption enabled: boolean. Describes whether this field will be encrypted when the document is encrypted with one or more secret (symmetric) keys.

Sign enabled: boolean. Describes whether this field should be covered by the signature under the following conditions:

- when mailed, if the sender uses signing.
- when saved, if the field occurs in a controlled section, and if the user modified the document in any way, and *could have* modified this section.

Example Fragment:

```
Field gross_salary
  encrypt_enable true
  sign_enable true
EndField
```

6.2.9 Documents

```
<doc> = Document <Identifier> [ <pattern>]
           [Signed <userName>]
           [Keys <keyList>]
           [Readers (<principalList>|All_readers_and_above)]
           [Authors <principalList>]
            ( <section> |
              <field>  )*
          EndDocument
```

Attributes:

Signed: user name. Describes the user who signed this whole document (as when mailed).

Keys: list of identifiers. Names of the keys that were used to encrypt the fields of this document that are encryption enabled.

Readers: list of principals or **All_readers_and_above.** Those who can read this document. If present, this list is intended to provide a per-document restriction on reading.

Authors: list of principals or "creator." Those who can edit this document if they have at least Author access to the database. Otherwise, they would need at least Editor access to the database to edit the document.

Composed of:
Sections, Fields

Example Fragment:

```
Document personel_folder
  Keys personel_dept_key
  Readers All_readers_and_above
  Section salary_approval
         Sign_enable True
  EndSection
EndDocument
```

6.2.10 Users

This kind of component is intended to be a placeholder for an actual person in the organization. The specification should not give a person's name, but rather name the role or place in the organization. In the form of a pattern, a user component represents a collection of people with similar security attributes.

Users and user patterns are the components in the Lotus layer that correspond to classes of users in the abstract layer.

```
<user> = User <Identifier> [Pattern]
           [Has Keys <keyList>]
           [CrossCert <OrgList>]
           [InGroup <GroupList>]
         EndUser
```

Attributes:

Keys: list of identifiers. The names of the keys that this user holds.

Cross-certifications: list of identifiers. The names of the domains (or organizations) in which this user is cross-certified. Since this version of the language is intended to handle only one domain, this attribute has little effect.

Groups: list of identifiers. The names of the groups of which this user is a member.

Example Fragment:

```
User employee pattern
     InGroup employee_group    # note: not an exhaustive list
EndUser

User trusted_employee pattern
   instance of employee  #note: pattern specialization
   Has keys management_key_1
   InGroup  trusted_group  #note: additional group
EndUser
```

6.2.11 Groups

A group is a shorthand within Lotus Notes for a collection of users. For instance, when a group is mentioned in an ACL entry, all of the members of that group are given the stated access.

Membership in each group is controlled by a document in the Public Address Book that lists the members.

```
<group> = Group <Identifier> [Pattern]
             Members <principalList>
          EndGroup
```

Attributes:

Members: list of the members

Example Fragment:

```
Group accouting_group
   Members head_accountant, typical_accountant, VP_financial
EndGroup
```

6.2.12 Other

Navigators, views, and folders are not currently incorporated in the specification. These do not play a significant role in the security policy.

Agents are also not mentioned. These are relevant for security but are beyond the scope of the current language. Section 10 of this report discusses some of the security ramifications of agents.

7. Connection Between Conceptual and Lotus Layers

The basic connections that can be specified identify:

- what conceptual data may be located in the Lotus components.
- in which conceputal user classes do the users belong

One can then check whether the Lotus structure supports the conceptual access rules at the specified level of seriousness. Some of these checks could be automated, and this is discussed briefly in Section 8.

For each kind of Lotus component, one can associate conceptual data. The syntax is:

```
<containsDecl> = contains  (<Identifier>  | <IdentifierList>)
                 contains only  (<Identifier> | IdentifierList>)
```

The first form says that the component may contain the data specified by the identifier(s). These identifiers are the names of abstract classes of data, declared in the conceptual layer. The second form is slightly stronger and asserts that that document may contain data specified by identifier(s) and no other kind of data.

A group or a user pattern may "contain" members from some classes of users, as declared in the Conceptual Layer, and this is expressed with the same syntax. Less clear is the meaning of this syntax when the component is a user (not a pattern). In that case, the concrete user must be a member of the abstract class of users (or the union of those classes, if more than one is named).

Example

```
Document Doc
  Contains Financial_data
  Readers a,b,c
EndDocument
```

For users, groups, and their patterns, an additional syntax is introduced:

```
<implementsDecl> = implements <Identifier>
```

When this qualifier is added to a group or user component, the described user or group of users is intended to correspond exactly to the class of users declared in the conceptual layer. That is, all members of the abstract class have concrete representatives described by this component, and all people described by this component are in the class.

The specification of seriousness provides another means of relating abstract and concrete notions. This is done by providing informal explanations for both the conceptual and the Lotus layer meanings of a seriousness identifier.

Another important kind of association between the Conceptual and Lotus layers is the relationship between functions and parameters. As discussed in section 6.1.3.1, a parameter in a pattern (in the Lotus layer) may correspond to a function in the conceptual layer. If the author has declared both a parameter and its corresponding function, then this relationship may be noted with the "is function" part of the parameter syntax:

```
<parameter> = parameter <paramName>
                  of kind <Kind>
                 [ in component <component_name> ]
                 [ is function <Identifier> ]
```

The identifier is the name of the function, as declared in the conceptual layer (see section 5.3).

8. User Interfaces, Analysis Methods and Tool Support

8.1 User Interfaces

We have developed a formal language that could be used as the basis for a set of possible tools for building a policy specification. We do not expect that the typical application developer will want, or need, to directly use the formal language. For example, to define the roles in the formal language, one might want:

```
UsersByJob defined as {Administrator, Manager, Programmer}
```

The user may be able to express this by using a dialog box with three list controls, which can be used to specify the choice of the object to be factored, the factorization operation, and the elements for that factorization.

This report does not cover the details of user interface design.

8.2 Analysis Methods

The language has been designed to facilitate checking conceptual policy against Lotus design.

Probably the most useful checks are those that determine if the access mechanisms specified in the Lotus part of the language comply with the conceptual rules. These checks could apply to any data containers in the Lotus part of the language that are specified as having conceptual data. The check should take into account controls at all the component layers. The check should also see if the seriousness criteria match the strength of the mechanisms.

There are also a variety of checks that can be performed to see if the specification is properly formed, including

- Type checking to make sure data and user elements are properly combined;
- Checking whether subset relations are compatible;
- Possibly checking if cardinality (size of sets) properties are consistent; and
- Whether patterns are being used properly (instances and parameters).

Conceptual consistency checks include:

- Conflicts between positive and negative conceptual access rules
- Conflicts between conceptual and Lotus layer rules (as described above)

Completeness analysis:

- Analysis of partial data specifications to show what relationships are not fully defined
- Determination of completeness of the data containers to conceptual data connections

8.3 Tool Support

Some of the analysis methods described in section 8.2 could have automated support. Tool design is not covered in this report.

9. An Example

We now discuss a simple example that illustrates the usefulness of our approach and language to model security policies. The example focuses on the security policy of a small organization, which we call ACME Corporation. We first articulate the security policy as it pertains to routine daily and monthly activities in a natural language (English) style. We then map these policy statements into expressions at the conceptual layer. These are then refined further into expressions at the Lotus layer. Finally, we specify the connections between these layers.

We have prototyped this example in Lotus Notes. This specification method was useful in designing the security prior to the application development phase.

9.1 Security Policy for ACME Corporation

ACME Corporation is a small research and consulting firm. The corporate structure and chain of command consists of the president, followed by vice-presidents, and directors. The other administration staff includes the personnel-manger, accountant, clerks and executive assistants. Technical staff includes project leads, group leads and junior technical staff members.

9.1.1 Administrative Activities

Daily Timesheet

- Employees need to access and modify timesheets, but unless explicitly authorized, an employee should have access to only his timesheet. However, the management group and company accountant may have read (access) to any timesheet at any time.
- All inputs to a timesheet including corrections to existing entries should be audited.

Monthly Payroll

- The company accountant should have read access to all employee timesheets. Any corrections to a timesheet required for accounting will have to be done by the appropriate employee.
- Only the company accountant is allowed to create a new timesheet on a monthly basis and modify certain fields, such as accrued vacation hours.
- As mentioned before, management group may have read access to all timesheets at any time.

Monthly Financial Report

- The accountant should have read and write access to monthly financial statements for projects as well as the entire company, and have the ability to create financial reports.
- The management group should have read access to all financial reports.
- Project Leads should have read access to the financial reports of all projects they are leading.

Update Personnel Information

- Certain admin staff, such as personnel clerks, may update general information of personnel such as address, telephone numbers, and job title.
- Only the company's personnel manager can create, read and modify sensitive personnel data, such as salaries, benefits and reviews.
- Senior management may read sensitive personnel data at any time, but they are not allowed to modify it.

9.1.2 Technical Activities

Monthly Project Report

- Only the senior-technical staff associated with a project may create or modify a project's report.
- All technical staff and management may read monthly project reports.

- Certain admin staff, such as the executive assistants to the President, VPs, and directors, may read and print project reports.

Writing Technical Proposals and Technical Reports

- Senior-technical staff may create, access, and modify technical proposal information at any time.
- All rights to Junior-technical staff for creating technical proposal documents and modifying existing proposal documents would have to be granted by senior-technical staff.
- The Management group may read technical proposal information at any time;
- Senior-management may modify technical proposal information only after approval from appropriate senior-technical staff.

9.2 Conceptual Layer Specifications

We now discuss the specification at the conceptual layer.

9.2.1 User Categories

We list the user categories below. Note that every user is considered an employee of the company.

```
Users Employee;
Users Technical-staff;
Users Admin-staff;
Users Employee partitioned by {Admin-staff, Technical-staff};
User President subset Admin-staff;
Users VP subset Admin-staff;
Users Director subset Admin-staff;
User Accountant subset Admin-staff;
User Personnel-manager subset Admin-staff;
Users Clerk subset Admin-staff;
Users Personnel-clerk subset Clerk;
Users Executive-assistants subset Admin-staff;
Users Project-lead subset Technical-staff;
Users Group-lead subset Technical-staff;
Users Management defined as Union {Project-lead, President, VP,
   Director};
Users Senior-management defined as union {President, VP};
Users Senior-technical defined as union {Project-lead, Group-lead};
Users Junior-technical defined as Difference {Technical-staff,
   Senior-technical};
Users No-modify-sensitive-personnel-data as Difference {Employee,
   Personnel-manager}
Users No-read-sensitive-personnel-data as Difference {Employee,
   Senior-management}
```

9.2.2 Data Categories

We now list the data categories. Every piece of data is considered to belong to the set Corporate-info.

```
Data Corporate-info partitioned by {Tech-data, Admin-data};
Data Tech-proposals;
Data Project-reports;
Data Project-tech-notes;
Data Personnel-details;
Data Personnel-salary;
Data Timesheets;
Data Accountant-data subset Timesheets;
Data User-data subset Timesheets;
Data Cost-proposals;
Data Project-finance-data;
Data Financial-summaries;
Data Corporate-finance-data;
Data Tech-data defined as union {Tech-proposals, Project-reports,
   Tech-notes};
Data Admin-data defined as union {Personnel-data,
   Corporate-finance-data};
Data Proposal-data defined as union {Tech-proposals, Cost-proposals};
Data Personnel-data partitioned by {Personnel-details,
   Personnel-salary,
Personnel-benefits, Personnel-reviews, Timesheets};
Data Corporate-finance-data defined as union {Project-finance-data,
   Cost-proposal};
Data Project-data defined as union {Project-reports,
   Project-tech-notes, Project-finance-data};
Data Sensitive-personnel-data defined as union {Personnel-salary,
   Personnel-benefits, Personnel-reviews};
Data Sensitive-corporate-data defined as union {Financial-summaries,
   Sensitive-personnel-data};
```

9.2.3 Activity Specification

```
Activity Daily-timesheet
   required
      {Accountant, Timesheet, Create}
      {Accountant, Accountant-data, Edit}
      {Employee, User-data, Edit}
      {Management, Timesheet, Read};

Activity Monthly-payroll
   required
      {Accountant, Timesheet, Create}
      {Accountant, Accountant-data, Edit}
      {Employee, User-data, Edit}
      {Management, Timesheet, Read};

Function Assign-Lead from Project-lead to Project-finance-data;
```

```
Activity Monthly-financial-reports
   required
      {Accountant, Project-finance-data, (Read, Edit, Create, Delete)}
      {Accountant, Financial-summaries, (Read, Edit, Create, Delete)}
      pl:Project-lead {pl, Assign-Lead(pl), Read}
      {Management, Project-finance-data, Read}
      {Management, Financial-summaries, Read}

Activity Update-personnel-information
   Required
      {Personnel-clerk, Personnel-details, (read, edit)}
      {Personnel-manager, Personnel-details, (create, read, edit)}
      {Personnel-manager, Sensitive-personnel-data,
                                            (create, read, edit)}
      {Senior-management, Personnel-details, read}
      {Senior-management, Sensitive-personnel-data, read}

Factor Privacy-of-sensitive-personnel-data
Denies
   {No-modify-sensitive-personnel-data, Sensitive-personnel-data, Edit}
         seriousness Very-high-privacy
   {No-read-sensitive-personnel-data, Sensitive-personnel-data, Read}
      seriousness Very-high-privacy;
```

We see that the access units in the factor are accompanied by the seriousness identifier Very-high-privacy. In ACME corporation, the Very-high-privacy seriousness calls for the following policy safeguards and organizational controls to be observed.

1. Under no circumstances can any security rule associated with this level of seriousness be ignored. There are no exceptions.
2. There must be yearly reviews to ensure that mechanisms of the appropriate strength are put in place and are verified to enforce (1).

At the Lotus layer, any data associated with a seriousness indicator of Very-high-privacy must be encrypted. In particular, the data must be encrypt-enabled in Lotus databases. The distribution of keys to decipher this data must be carefully controlled.

9.3 Lotus Layer

Before we develop the Lotus layer specification, we informally map the abstract classifications of data at the conceptual layer to the data types supported by Lotus. These include forms, documents, and sections.

9.3.1 Informal Description of Lotus Documents

9.3.1.1 Proposal-data

Proposal data can be represented as a Lotus Notes document with two sections.

- Section 1 contains technical proposal
- Section 2 contains cost proposal

9.3.1.2 Personnel-data

Personnel-data for an employee can be represented as four sets of documents.

Personnel-details. This document captures all the basic information about an employee, such as name, address, social security number, job title, and salary.
Personnel-benefits. This document summarizes the employee's benefits, such as retirement savings plan, profit sharing, and insurance options.
Personnel-reviews. This document summarizes all the annual reviews of the employee.
Personnel-timesheets. This set of documents comprise an employee's monthly timesheets.

9.3.1.3 Project-data

The project data for every project comprises the following sets of documents.

Project-reports. This set contains monthly project reports.
Project-tech-notes. This set contains technical notes made by the technical members on a project.
Project-financial-statements. This contains monthly financial statements and collectively represents the project's financial data.

9.3.2 Lotus Layer Specification

We start by giving the Lotus Layer specification for databases, forms, documents, sections, and fields.

```
Organization Acme_Corporation
   OrgUnit Head_office
      Server Acme_main_server
         Database Proposals-db
            Roles {Technical-staff, Junior-technical,
                   Senior-technical,  Management, Senior-management}
            ACL prefix {Senior-technical,Author}
                       {Junior-technical, Readers}
                       {Management,Readers}
            Form Proposal-info-form
               Section Technical-section
               Section Cost-section
            EndForm
         EndDatabase

         Database Personnel-db
            Roles {Clerk, Personnel-manager, Mangement}
            ACL prefix {Personnel-manager, Author}
                       {Accountant, Author}
                       {Personnel-clerk, Reader}
            {Senior-management, Reader}
            {Employees, Editor}

   Form Personnel-details-form
      DefaultReaders {Personnel-clerk, Accountant,
         Senior-management}
      Creators {Personnel-manager}
      Keys {salary-key}

      Field Salary
         Encrypt_Enable true
      EndField
   EndForm

   Form Personnel-benefits-form
      DefaultReaders {Accountant, Personnel-manager,
         Senior-management}
      Creators {Personnel-manager}
   EndForm

   Form Personnel-reviews-form
      DefaultReaders {Personnel-manager, Senior-management}
      Creators {Personnel-manager}
   EndForm

   Form Personnel-timesheets-form
      Creators {Accountant}
      ContainsAuthorsField

      Section User-hours-input
         Editors "Employee whose name is on the timesheet"
         Field Hours-per-day pattern
            Sign-enabled true
         EndField
      EndSection

      Section Accountant-input
         Editors {Accountant}
         Field Vacation-available
            Sign-enabled true
         EndField
      EndSection
   EndForm
EndDatabase

Database Projects-db
   ACL prefix {Senior-technical, Author}  {Accountant, Author}
      {Executive-assistants, Reader}  {Technical-staff,Reader}
      {Management, Reader}
   Form Project-reports-form
      DefaultReaders {Executive-assistants, Technical-staff,
          Management }
      CreateAccessList {Senior-technical}
   EndForm
```

```
            Form Project-tech-notes-form
               DefaultReaders {Executive-assistants, Technical-staff,
                  Management }
               Creators {Senior-technical}
            EndForm

            Form Project-financial-statements
               DefaultReaders {Project-lead, Management}
               Creators {Accountant}
            EndForm
         EndDatabase
      EndServer
   EndOrgUnit
EndOrganization
```

These specifications express the Lotus layer specifications for databases, forms, documents, sections and fields.

We conclude this section by discussing the additional specification statements that give the connection between the Conceptual and Lotus layers. These statements should be combined with the other parts of the Lotus layer specifications.

Let us start first with data. We notice that all personnel data at the conceptual layer is captured in the personnel database at the Lotus layer. This is expressed by

```
Personnel-db
   contains only Personnel-data
```

Similarly, we can write the specifications for the proposal and projects data.

```
Proposals-db
   contains only Proposal-data
Project-db
   contains only Project-data
```

Finally, we need to specify the connection between users, groups, and roles among the different layers. For simplicity, we have assumed a one-to-one mapping between the identifiers used for the users, groups, and roles in the conceptual and Lotus layer specifications. We list some of user and group definitions below.

```
User technical-staff pattern
   implements technical-staff
EndUser
User Admin-staff pattern
   implements Admin-staff
EndUser
User Accountant pattern
   implements Accountant
EndUser
```

```
User Personnel-manager pattern
   implements Personnel-manager
   Has Keys salary-key
EndUser
User Personnel-clerk pattern
   implements Personnel-clerk
   InGroup Clerk
EndUser
User Project-lead pattern
   implements Project-lead
   InGroup Management
EndUser
User President
   implements President
   InGroup Management, Senior-management
EndUser
User VP pattern
   implements VP
   InGroup Management, Senior-management
EndUser
User Director
   implements Director
   InGroup Management
EndUser

Group Management
   implements Management
   Members Project-lead, President, VP, Director
EndGroup
Group Senior-management
   implements Senior-management
   Members President, VP
EndGroup
```

This concludes the Lotus layer specifications.

10. Possible Extensions to Lotus Notes

10.1 Existing Lotus Security

Lotus provides a rich and flexible set of security controls. One major area in which extensions may be useful is support for complex security policies. This could be done either by adding extensions to Lotus Notes or through third-party software. For this effort, we have not examined third-party software. Also, our discussion applies to version 4.1 of Lotus Notes.

10.2 Some General Areas

Lotus Notes has above average security controls. Yet, even this technology will not fully enforce every kind of security policy.

Areas for which there might be possible enhancements (based on Lotus Notes 4.1 documentation) include

- version control,
- transaction support,
- real-time and priority trade-offs, and
- auditing support.

Note, some of these items are being addressed in new updates to Lotus Notes and Domino.

10.3 Dynamic Access Controls and Active Security

Lotus Notes developers can potentially incorporate security policies that reflect how the data is evolving (see section 4.5.1). However, the Lotus mechanisms available to support such policies can be difficult to employ properly if the policy is complex. (We examined Lotus Notes 4.1, and not third party software.) In this section, we describe a useful type of complex, security policy, describe the Lotus mechanisms for enforcement, and then describe potential Lotus Notes enhancements. In particular, we examine policies for the active runtime management of security tasks. We call such policies, *active security* policies.

The central aspect of active security is the coordination, management and enforcement of security in accordance with how a task, or group of tasks, has progressed. Active security notions are important for application environments such as collaborative work, groupware, and workflows, as they can more accurately express security principles such as least privilege and need-to-know.

From an access control standpoint, active security can be provided by dynamically manipulating access control information so that accesses can be denied and granted in accordance with policies that apply to the current or emerging context of a task. In essence, active security enforcement requires the ability to provide support for just-in-time permissions and to dynamically activate and deactivate permissions in a policy-driven manner.

We illustrate the concept of active security with an example.

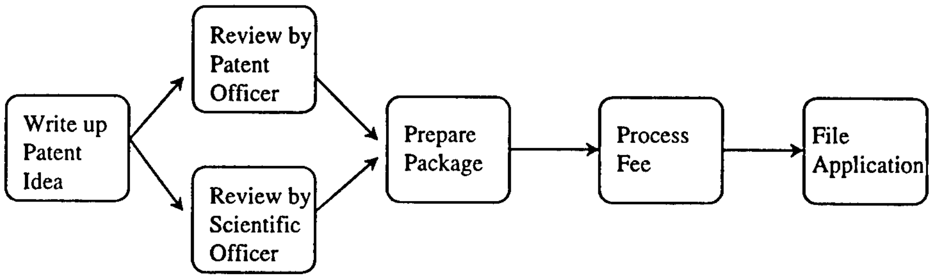

Figure 3: A patent processing workflow

Figure 3 shows the typical steps involved in a workflow for writing and processing a patent application internally within an enterprise. The first activity involves various members of a research group writing the idea for the patent. These authors will require read and write permissions to a set of electronic documents. Once the writing is completed, these documents are forwarded to the patent and scientific officers for review. During reviews, the authors should not be allowed to make any modifications. Thus, the write permission of the authors should be turned off during the review activity. Once the reviewers are done with the reviews, the authors should be allowed to incorporate the comments and suggestions of the reviewers. This would require that the authors have write permissions once again to relevant documents. Eventually, the documents will be sent to the art department for finishing touches. At this phase, the authors should not be allowed to make any more modifications and thus their write permissions have to be turned off again. As the workflow progresses to other stages, such as "Process Fee" and "File Application," permissions to various users (billing clerks and filing clerks) would have to be turned on and off.

The above example illustrates the need to dynamically manipulate access control permissions, depending on how far a collaborative activity has progressed. Let us now see how this can be done in Lotus notes.

10.3.1 Active Controls at the Database Level

To implement the patent processing example in Lotus, one could create a patent applications database that stores the relevant documents that comprise patent applications. A first approach to providing active access controls in Lotus notes would involve the manipulation of ACLs on the database. Such an approach can provide access controls only at a very coarse level, as opposed to controls possible at the level of individual documents (we shall discuss this possibility later).

Every ACL of a Lotus database is considered an object instance of a Lotus class called NotesACL. The NotesACL class provides appropriate methods to manipulate ACLs. These methods include adding, deleting, and renaming roles, as well as retrieving access control entries within an ACL.

Figure 4 illustrates an approach to providing active access controls in Lotus with ACLs. It uses four essential features in Lotus Notes:

1. *Agents.* These are lightweight processes that can perform specific functions.
2. *Event monitoring.* This provides the ability to recognize the occurrence of events.
3. *Scripts.* These are programs written using the LotusScript programming language and can be attached to various objects such as agents.
4. *ACL manipulation under program control.* This feature in Lotus Notes allows the ACLs of databases to be manipulated through LotusScript programs (scripts).

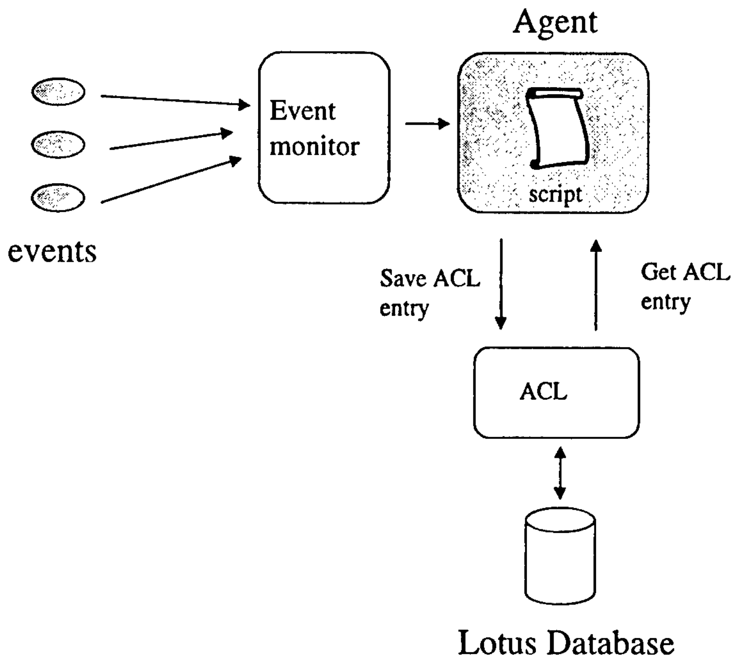

Figure 4: Active access controls on Lotus databases using agents and scripts

The central idea is to use script-based agents to monitor events and to take appropriate security actions. An agent will take appropriate action by activating a stored script. This script encompasses the security policy that is used to dynamically manipulate the ACL of a Lotus database. Lotus provides for the ability to recognize the occurrence of a variety of events, such as the receipt of e-mail, modifications of documents, and the activation of certain buttons.

To incorporate active access controls in the patent processing workflow example, we can use agents that recognize the termination of tasks. Once a task has terminated, the agent can manipulate the ACLs of the database so that the appropriate access controls are set before the next task in the workflow is started. The actual signaling of the termination of a task may be done in several ways. If the user is to signal termination, the simplest way to do this might be for the user to press a "finish" button on a document. The necessary input information to determine which users may be granted or denied further permissions as a result of this action may be obtained in several ways. For example, in the patent-processing example, an author may designate a particular scientific officer to review the application. In this case, the agent will have to retrieve the officer's name or id from the user interface and pass it to the ACL manipulation methods. This will result in the scientific officer getting access to the patents applications database. Alternatively, the determination of the users or roles that are to be granted or denied permissions may be a function of where the documents are routed next. Thus, when the patent application workflow advances to the Process Fee and File Application phases, a user in the roles of billing-clerk and filing-clerk may be given access to the patent application documents.

10.3.2 Active Controls at the Document Level

In the previous section, section 10.3.1, we addressed active security at the coarse level of a Lotus Notes database. Lotus also provides mechanisms for manipulating access controls to individual forms and documents and these can be exploited to provide active access controls at the document level. This is done by creating the following fields on a form:

1. **Readers field**. This field specifies who can read documents created with the form.
2. **Authors field**. This field specifies users who can edit documents created with the form.

Lotus Notes 4.1 does not allow for the manipulation of document reader and author fields through scripts and agents. Instead, to implement active access controls one needs to specify the fields as being of type "computed." This allows an application developer to specify commands to modify the above fields using the Lotus formula language. However, it is important to note that the Lotus formula language is not as expressive or powerful as the LotusScript language. The LotusScript language is a full-fledged object-oriented programming language while Notes formulas are expressions that provide limited control logic and some basic facilities for the binding of variables. In any case, it is important to realize that the read and author access lists cannot override a database access control list; they can only refine it.

10.3.3 Disadvantages of Implementing Active Controls in Lotus Notes

These approaches for implementing active security have some inherent drawbacks. Some of these include the following:

- The security policy is buried and scattered in Lotus scripts and formulas.
- The approach to specifying and incorporating security policy is at a low level.
- There is no central point of administration for active security.

These collectively result in an approach that is error prone and hard to maintain. As application developers specify security at the low level of Lotus scripts and formulas, they can not be sure if they have captured high-level security policies accurately, completely, or consistently. As there is no single point of administration, a change in one script or formula may affect other documents and databases through agent-based actions. There is no central point of traceability to changes in policy made through modifications of scripts and formulas.

10.3.4 Possible Extensions to Lotus

The disadvantages associated with implementing active security in Lotus Notes open opportunities for building extensions and related high level security tools on top of Lotus Notes.

It would be useful to extend the policy specification language to express active security requirements for tasks. Such a language should allow an application developer to specify for each task the following:

- The start-up, termination, and other events that could occur during the life of the task.
- The relevant databases and documents that are to be accessed by users and agents during the lifetime of the task.
- Associations between events (or states) and permission granting and denying functions to all documents and databases pertinent to the task.

Tools such as policy editors for the language can be built to incorporate general business process and workflow modeling methodologies, such as those described in Chapter 11 of the Lotus Notes Application Developers Guide.

A mapping from sentences expressed in the specification language to lower level statements in Lotus scripts and formulas might be provided. This could be done in several ways with varying degrees of automation and sophistication.

- Use of generic and parameterized scripts that can be manually specialized by application developers to express individual policies.
- The use of a compiler or translator to provide the automated mapping from the high level language to Lotus Notes scripts and formulas.

11. Conclusions

The policy language described in this report provides a means by which an application developer can specify both an abstract access control policy and a more concrete description of security attributes of Lotus objects. Additionally, the seriousness of enforcing access can be specified. This permits a more organized and careful way of specifying security. Since Lotus Notes provides a rich environment for controlling security, this specification method should reduce the possibility that certain security situations are not covered, or that inappropriate controls are used.

12. Recommendations

To make this approach more accessible to application developers, it is recommended that tools be developed to aid the developer in building and checking the specification.

It would also be desirable to have sample example specifications that correspond to applications and templates that will be commonly used.

References

Lotus Notes – Application Developer's Guide (release 4)
Lotus Notes – Programmer's Guide, Part 1 (release 4)
Lotus Notes – Programmer's Guide, Part 2 (release 4)

IBM Red Book on Lotus Notes security (draft), www.redbooks.ibm.com/SG244848

☆U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81083

MISSION OF AFRL/INFORMATION DIRECTORATE (IF)